REMARKS

Claims 1, 7, 25, 32, and 33 have been amended. Claim 2 has been cancelled. No new claims have been added, Consequently, claims 1, 3-7, 24-25, and 32-33 are pending.

The specification and claims 1-7 and 24 stand rejected under 35 U.S.C. § 112, first paragraph, as allegedly failing to comply with the written description requirement. More specifically, the Office Action alleges that the claim limitation of "a prescribed change in a state of said scheduling policy" (claim 1) and enabling/disabling "context switching between threads when said flag is set to correspond to the" absence/presence "of context switching" (claim 24) are not supported by the specification.

The present invention is directed to an apparatus and method for granting and revoking exclusive access rights to one or more threads executing on a processor based system. A thread is a basic unit of execution in a computer system. Processes are groups of at least one thread. In a multi-tasking computer system, a scheduler component of the operating system is used to context switch the execution state of the computer system among the plurality of threads. In the prior art, exclusive access is granted to a thread through the use of traditional mutual exclusion mechanisms, such as semaphores.

The present invention discloses an apparatus and method for granting exclusive access to a resource by controlling thread scheduling. A scheduler in accordance with the principles of the present invention monitors the state of a scheduling flag, which is typically in a "second" state. When the scheduling flag is in the second state, the scheduler performs context switching of the processor's execution state among the plurality of threads on the computer system in accordance to a scheduling policy. However, if the scheduling flag is in a "first" state, the scheduler does not perform context switching as long as the scheduling flag is in the first state. An application programming interface (API) is provided to permit threads to call functions which reads or writes to the scheduling flag. See, e.g., page 11, lines 13-25 (describing the scheduler's use of the scheduling flag and corresponding API functions).

The Examiner's attention is directed to, for example, page 11, lines 13-25 which, as amended, states:

> The program control apparatus in accordance with an aspect of the present invention controls program execution in a computer system in which threads are switched in accordance with a scheduling policy by a scheduler. The program control apparatus includes a first unit responsive to a predetermined first application program interface call from a thread for setting a prescribed flag to either one of first and second state; a second unit for setting, after the flag is set to one of the aforementioned states, the flag to the other one of the first and second states upon detection of a prescribed change in the state of the computer system,; and a third unit responsive to a second application program interface call paired with the first application interface call from the aforementioned thread, for returning a value indicating the state of the flag to the aforementioned thread.

The Examiner's attention is further directed to, for example, page 12, lines 7-18, which state:

> Preferably, the first unit includes a unit responsive to an application program interface call requesting start of detection of presence/absence of a context switch from a thread for setting a flag indicating presence/absence of a context switching to a state corresponding to absence of a context switching; the second unit includes a unit for setting, after the flag is set to the state corresponding to the absence of a context switching and a scheduler switches a context, the flag to a state corresponding to presence of a context switching; and the third unit includes a unit responsive to an application program interface call requesting termination of detection of presence/absence of a context switching from the aforementioned thread, for returning a value corresponding to the state of the flag to the aforementioned thread.

It is respectfully submitted that the above quoted paragraphs do in fact disclose a scheduling policy having at least two states which correspond to the sates of the flag. In answer to the Examiner's assertion that "a scheduling policy generally describes how a thread is scheduled for execution relative to the other threads" and the associated question of "[i]t is unclear to the [E]xaminer as to how a scheduling policy can have a state," it

should be noted that "scheduling policy" refers to the process performed by the scheduler in deciding which thread to next execute, and each process executable by a computer system has an equivalent state machine. Thus, a scheduling policy must necessarily have states corresponding to that state machine. In the context of the invention, these states include the states of enabled context switching and disabled context switching.

Finally, with respect to the Examiner's comments on page 2 of the Office Action stating that "applicant discloses throughout the specification that context switch can occur while the flag is set to correspond to the absence of context switching," in which the Examiner cites to page 18, lines 5-16, page 23, line 7-20, page 34, lines 4-20; page 35, lines 17-25, page 36, lines 15-20, page 37, lines 18-28, and Figs. 15A, 15B, 19, and 20, it should be noted that this is directed to an alternate embodiment, described, at, for example, page 34, lines 4-29. More specifically, the process described at page 34 lines 4-29 are directed to a method of mutual exclusion where a thread or process requiring mutual exclusion to a resource executes an API call "API#A" prior to execution and an API call "API#B" immediately after execution. The second API call produces a return value indicating whether a context switch occurred during the time between the two API calls and if so, any thread/process executed between the two API calls has its process space discarded and is rescheduled for execution, thereby achieving mutual exclusion without actually locking the system. Compare, e.g., page 11, line 26 – page 12, line 1 with page 12, lines 2-6).

Accordingly, the rejection under 35 U.S.C. § 112, first paragraph with respect to claim 1 (and depending claims 2-7) should be withdrawn. Further, with respect to claim 24, the second of the above quoted paragraph clearly indicates that one state of the flag corresponds to the presence of context switching while the other state corresponds to the absence of context switching. Accordingly, the rejection under 35 U.S.C. § 112, first paragraph with respect to claim 24 should be withdrawn.

Claim 1 stands rejected under 35 U.S.C. § 102(b) as being anticipated by Record (U.S. Patent No. 5,335,383). Claims 2, 3, 24, and 32 stand rejected under 35

U.S.C. 103(a) as being unpatentable over Record in view of Lehr (U.S. Patent No. 5,898,873). Claim 4 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over Record, Lehr, and Borkenhagen (U.S. Patent No. 6,212,544). Claim 5-6 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Record, Lehr, and Ellis. Claims 7, 25, and 33 stand rejected under 35 U.S.C. § 103(a) over Record and Toutonghi (U.S. Patent No. 5,842,016). Claim 24 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over Record, Lehr, and Song (U.S. Patent No. 6,061,711). These rejections are respectfully traversed.

As noted above, in accordance to one aspect of the present invention, context switching between threads may be controlled (i.e., enabled or disabled) based upon the state of a flag, and the state of the flag may be set by calling API routines. Accordingly, independent claims 1, 24 and 32 each recite:

> ... disabling context switching between threads when said flag is set to correspond to absence of context switching ... enabling context switching between threads when said flag is set to correspond to presence of context switching ...

According to another aspect of the present invention, the API further include additional functions which is used to establish a flag taking a state which reflects whether a region of memory has been written, and which is used to read the value of the flag. In this manner, one thread can monitor whether another thread has written the memory address range of interest. If the memory address range is carefully chosen, the ability to know that the memory address range was not written can be used to ensure that access to a resource was exclusively to a thread, while the ability to know that the memory address was written to could be used to take corrective action. See specification, p. 13. Accordingly, independent claims 25 and 33 recite:

> ... setting said flag to a state corresponding to presence of a data write when there is a data write to said designated memory area and setting

said flag to another state when there is no data write to the designated memory area ...

Record is directed to an event management system for a computer operating system. More specifically, Record teaches a general purpose event management system which includes an event manager (e.g., Fig. 1, manager 26) which interacts with event definitions (e.g., Fig. 1, definitions 16a-16d), event monitors (e.g., Fig 1, monitors 28a-28d), and event handlers (e.g., Fig. 1, handlers 14a-14d). Application programs executing on a computer system with the event management system of Record may define a variety of events (e.g., completion of a function, reaching specific checkpoints in computer code, application error exceptions), and have the event management system call an appropriate event handler if an event occurs. See column 6, lines 4-67; Figs. 5-6. Record therefore teaches a flexible event management system. However, Record fails to teach or suggest the above quoted limitations of the independent claims.

Song is directed to a method for achieving efficient context saving and context restoring in a multi-tasking computing environment including a processor and a co-processor. The Office Action alleges that Song discloses disabling context switching when the CSE flag is not set and enabling context switching when the CSE flag is set. The Office Action alleges that this is taught at column 10, lines 21-35, column 10, line 55 – column 11, line 5; and column 11, lines 40-46. However, column 10, lines 21-35 merely discloses that the processor 202 writes to a 32-bit vector processor interrupt mask register 208 (i.e., register VIMSK) in order to request a context switch. Further, column 10, line 55 – column 11, line 5 merely discloses that the co-processor is used, while processing the VCCS instruction, to determine whether the processor requested a context switch by reading the VIMSK register. Finally, column 11, lines 40-46 explains the operation of the VCCS instruction in relation to the pseudo-code reproduced at column 11, lines 30-39. More specifically, Song discloses that the CSE bit of the VIMSK register is examined to determine whether a processor 202 requested a context switch. That is, Song does not disclose or suggest a mechanism which changes the state of a scheduling policy such that context switches are either enable or disabled. The portion of Song cited by the Office Action merely relates to how a processor/co-processor pair can be used to detect a request

for a context switch. Song therefore fails to teach or suggest disabling/enabling context switching based on the state of a flag which can be controlled via API routines, as required by claim 24.

Lehr is directed to a computer execution trace analysis tool. Computer execution traces are logs which report the execution activity of a computer system. Programmers may need to use execution traces while debugging to analyze exactly what a piece of computer code is doing. One issue which arises when interpreting traces is that in multi-tasking operating systems, one process or thread may be context switched for another process of thread, in accordance with the policy of a scheduler component of the operating system. In some computer systems, context switches are easily detected, while in others, especially operation systems which depend upon "cooperative multitasking" (e.g., Microsoft Windows), context switches are less predictable because a context switch can only occur when, for example, a Windows program calls an API routine to examine the event queue. Although Lehr understands the concept of a context switch, Lehr fails disclose or suggest an ability to enable or disable context switches by a scheduler, and controlling the scheduler using a flag whose state can be changed by calling an API routine.

Toutonghi discloses a garbage collection system in a computer system supporting threads. Toutonghi discloses a system which does not actually perform garbage collection (even after garbage collection has been "initiated") until each thread exits each resource accessing section. This minimizes the amount of time each thread is negatively impacted by the garbage collection process. See column 2, lines 32-63. Toutonghi discloses, *inter alia,* two API routines: DisableGarbageCollection and EnableGarbageCollection and these two routines respectively set and reset a DisableGC flag. However, contrary to the assertion made in the Office Action, Toutonghi does not disclose or suggest "setting said flag to a state corresponding to presence of a data write when there is data write to the designated memory area." The EnableGarbageCollection API merely resets the DisableGC flag and the state of the DisableGC flag does not necessarily indicate whether a memory range has been written (although the garbage

collection process may write that region after the DisableGC flag has been reset). Further, claims 25 and 33 have been require the flag taking one state corresponding to the presence of a data write and another state corresponding to the absence of a data write. Thus, the teaching of a read or write access which sets a flag to a particular state no longer reads upon claims 25 and 33.

The Office Action additionally cites to Borkenhagen and Ellis for various teachings. However, neither Borkenhagen and Ellis, whether taken singly or in combination, discloses or suggests the above-recited limitations of the independent claims. Accordingly, independent claims 1, 24, 25, 32, and 33 are believed to be allowable over the prior art of record. Claims 2-7, which depend from claim 1, are also believed to be allowable over the prior art of record for these reasons and because the combinations defined in the claims are not shown or suggested by the cited references.

In view of the above, each of the presently pending claims in this application is believed to be in immediate condition for allowance. Accordingly, the Examiner is respectfully requested to pass this application to issue.

Dated: August 28, 2003

Respectfully submitted,

By_____

Thomas J. D'Amico
Registration No.: 28,371
Christopher S. Chow
Registration No.: 46,493
DICKSTEIN SHAPIRO MORIN & OSHINSKY LLP
2101 L Street NW
Washington, DC 20037-1526
(202) 785-9700
Attorneys for Applicant